

12 | 三分天下的容器：恰当选择，事半功倍

2020-06-02 罗剑锋

罗剑锋的C++实战笔记

[进入课程 >](#)



讲述：Chrono

时长 16:01 大小 14.68M



你好，我是 Chrono。

今天我要讲的是标准库里的一块“重地”：容器，它也是 C++ 泛型编程范式的基础。

不过在正式开讲之前，我先问你个问题：什么是容器？

你也许会说：**容器，就是能够“容纳”“存放”元素的一些数据结构。**

这个回答非常正确，而且说到了“点”上。



还记得计算机先驱的那句经典名言吗？**“算法 + 数据结构 = 程序。”**在 C++ 里，容器就是这个公式里面的“数据结构”。

所以，下面我就着重从数据结构的角度，来谈谈各种容器的区别、优缺点，还有如何选择最合适的容器。

认识容器

所谓的数据结构，就是数据在计算机里的存储和组织形式，比如堆、数组、链表、二叉树、B+ 树、哈希表，等等。

在计算机的发展历史上，众多“大牛”孜孜不倦地发明创造了这么多的数据结构，为什么呢？

因为没有一种数据结构是万能的、可以应用于任何场景。毕竟，不同的数据结构存储数据的形式不一样，效率也就不一样。有的是连续存放，有的是分散存放，有的存储效率高，有的查找效率高，我们必须依据具体的应用场合来进行取舍。

我想，你肯定已经学过这些数据结构了，也知道它们的实现原理，自己写也不是什么太难的事情。

但是，对于最基本、最经典的那些数据结构，你完全没有必要去“自己造轮子”，因为 C++ 标准库里的容器就已经把它们给实现了。

容器，其实就是 C++ 对数据结构的抽象和封装。而且，因为标准库开发者的功力很深，对编译器的了解程度更是远超你我，所以，容器的性能和优化水平要比我们自己写的好上几十倍，这一点你绝对不用质疑。

我们要做的，就是仔细品鉴标准容器这盘大餐，从中找出最合适自己口味的那道菜。

由于容器相关的资料已经有很多了，无论是看图书还是网站，都可以找到非常详细的接口文档，所以今天，我就不去罗列每个容器的具体操作方法了，而是把重点放在特性介绍上。掌握了这些特性，今后你在面临选择的时候，不用太纠结，就可以选出最适合你的容器。

容器的通用特性

你必须要知道所有容器都具有的一个基本特性：它保存元素采用的是“值”（value）语义，也就是说，**容器里存储的是元素的拷贝、副本，而不是引用。**

从这个基本特性可以得出一个推论，容器操作元素的很大一块成本就是值的拷贝。所以，如果元素比较大，或者非常多，那么操作时的拷贝开销就会很高，性能也就不会太好。

一个解决办法是，**尽量为元素实现转移构造和转移赋值函数**，在加入容器的时候使用 `std::move()` 来“转移”，减少元素复制的成本：

 复制代码

```
1 Point p; // 一个拷贝成本很高的对象
2
3 v.push_back(p); // 存储对象，拷贝构造，成本很高
4 v.push_back(std::move(p)); // 定义转移构造后就可以转移存储，降低成本
```

你也可以使用 C++11 为容器新增加的 `emplace` 操作函数，它可以“就地”构造元素，免去了构造后再拷贝、转移的成本，不但高效，而且用起来也很方便：

 复制代码

```
1 v.emplace_back(...); // 直接在容器里构造元素，不需要拷贝或者转移
```

当然，你可能还会想到在容器里存放元素的指针，来间接保存元素，但我不建议采用这种方案。

虽然指针的开销很低，但因为它是“间接”持有，就不能利用容器自动销毁元素的特性了，你必须要自己手动管理元素的生命周期，麻烦而且非常容易出错，有内存泄漏的隐患。

如果真的有这种需求，可以考虑使用智能指针 `unique_ptr`/`shared_ptr`，让它们帮你自动管理元素。建议你再仔细复习一下 [第 8 讲](#) 的内容，弄清楚这两个智能指针之间的差异，区分“独占语义”和“共享语义”。

一般情况下，`shared_ptr` 是一个更好的选择，它的共享语义与容器的值语义基本一致。使用 `unique_ptr` 就要当心，它不能被拷贝，只能被转移，用起来就比较“微妙”。

容器的具体特性

上面讲的是所有容器的“共性”，接下来我们再来看看具体容器的“个性”。

C++ 里的容器很多，但可以按照不同的标准进行分类，常见的一种分类是依据元素的访问方式，分成**顺序容器**、**有序容器**和**无序容器**三大类别，先看一下最容易使用的顺序容器。

顺序容器

顺序容器就是数据结构里的线性表，一共有 5 种：array、vector、deque、list、forward_list。

按照存储结构，这 5 种容器又可以再细分成两组。

连续存储的数组：array、vector 和 deque。

指针结构的链表：list 和 forward_list。

array 和 vector 直接对应 C 的内置数组，内存布局与 C 完全兼容，所以是开销最低、速度最快的容器。

它们两个的区别在于容量能否动态增长。array 是静态数组，大小在初始化的时候就固定了，不能再容纳更多的元素。而 vector 是动态数组，虽然初始化的时候设定了大小，但可以在后面按需增长，容纳任意数量的元素。

 复制代码

```
1 array<int, 2> arr;           // 初始一个array, 长度是2
2 assert(arr.size() == 2);    // 静态数组的长度总是2
3
4 vector<int> v(2);           // 初始一个vector, 长度是2
5 for(int i = 0; i < 10; i++) {
6     v.emplace_back(i);      // 追加多个元素
7 }
8 assert(v.size() == 12);     // 长度动态增长到12
```

deque 也是一种可以动态增长的数组，它和 vector 的区别是，它可以在两端高效地插入删除元素，这也是它的名字 double-end queue 的来历，而 vector 则只能用 push_back 在末端追加元素。

 复制代码

```
1 deque<int> d;               // 初始化一个deque, 长度是0
2 d.emplace_back(9);          // 末端添加一个元素
```

```
3 d.emplace_front(1);           // 前端添加一个元素
4                               // 形成双向链表
```

vector 和 deque 里的元素因为是连续存储的，所以在中间的插入删除效率就很低，而 list 和 forward_list 是链表结构，插入删除操作只需要调整指针，所以在任意位置的操作都很高效。

链表的缺点是查找效率低，只能沿着指针顺序访问，这方面不如 vector 随机访问的效率高。list 是双向链表，可以向前或者向后遍历，而 forward_list，顾名思义，是单向链表，只能向前遍历，查找效率就更低了。

链表结构比起数组结构还有一个缺点，就是存储成本略高，因为必须要为每个元素附加一个或者两个的指针，指向链表的前后节点。

vector/deque 和 list/forward_list 都可以动态增长来容纳更多的元素，但它们的内部扩容机制却是不一样的。

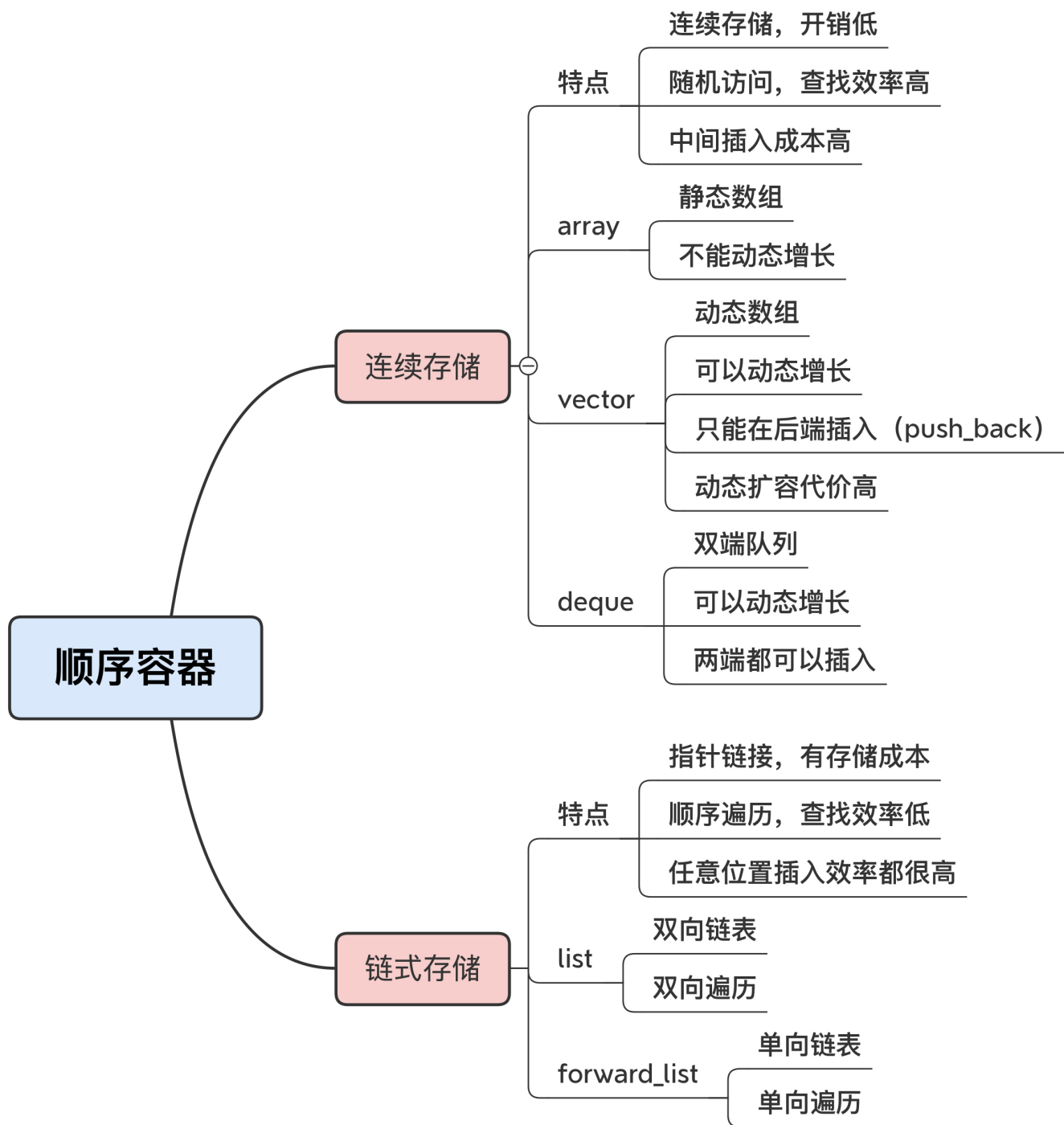
当 vector 的容量到达上限的时候（capacity），它会再分配一块两倍大小的新内存，然后把旧元素拷贝或者移动过去。这个操作的成本是非常大的，所以，你在使用 vector 的时候最好能够“预估”容量，使用 reserve 提前分配足够的空间，减少动态扩容的拷贝代价。

vector 的做法太“激进”，而 deque、list 的扩容策略就“保守”多了，只会按照固定的“步长”（例如 N 个字节、一个节点）去增加容量。但在短时间内插入大量数据的时候就会频繁分配内存，效果反而不如 vector 一次分配来得好。

说完了这 5 个容器的优缺点，你该怎么选择呢？

我的看法是，如果没有什么特殊需求，首选的容器就是 array 和 vector，它们的速度最快、开销最低，数组的形式也令它们最容易使用，搭配算法也可以实现快速的排序和查找。

剩下的 deque、list 和 forward_list 则适合对插入删除性能比较敏感的场所，如果还很在意空间开销，那就只能选择非链表的 deque 了。



有序容器

顺序容器的特点是，元素的次序是由它插入的次序而决定的，访问元素也就按照最初插入的顺序。而有序容器则不同，它的元素在插入容器后就被按照某种规则自动排序，所以是“有序”的。

C++ 的有序容器使用的是树结构，通常是红黑树——有着最好查找性能的二叉树。

标准库里一共有四种有序容器：set/multiset 和 map/multimap。set 是集合，map 是关联数组（在其他语言里也叫“字典”）。

有 multi 前缀的容器表示可以容纳重复的 key，内部结构与无前缀的相同，所以也可以认为只有两种有序容器。

因为有序容器的数量很少，所以使用的关键就是要理解它的“有序”概念，也就是说，**容器是如何判断两个元素的“先后次序”，知道了这一点，才能正确地排序。**

这就导致了有序容器与顺序容器的另一个根本区别，**在定义容器的时候必须要指定 key 的比较函数**。只不过这个函数通常是默认的 less，表示小于关系，不用特意写出来：

 复制代码

```
1  template<
2      class T                                // 模板参数只有一个元素类型
3  > class vector;                            // vector
4
5  template<
6      class Key,                             // 模板参数是key类型，即元素类型
7      class Compare = std::less<Key>        // 比较函数
8  > class set;                               // 集合
9
10 template<
11     class Key,                             // 第一个模板参数是key类型
12     class T,                               // 第二个模板参数是元素类型
13     class Compare = std::less<Key>        // 比较函数
14 > class map;                              // 关联数组
```

C++ 里的 int、string 等基本类型都支持比较排序，放进有序容器里毫无问题。但很多自定义类型没有默认的比较函数，要作为容器的 key 就有点麻烦。虽然这种情况不多见，但有的时候还真是个“刚性需求”。

解决这个问题有两种办法：一个是重载“<”，另一个是自定义模板参数。

比如说我们有一个 Point 类，它是没有大小概念的，但只要给它重载“<”操作符，就可以放进有序容器里了：

 复制代码

```

1 bool operator<(const Point& a, const Point& b)
2 {
3     return a.x < b.x;           // 自定义比较运算
4 }
5
6
7 set<Point> s;                   // 现在就可以正确地放入有序容器
8 s.emplace(7);

```

另一种方式是编写专门的函数对象或者 lambda 表达式，然后在容器的模板参数里指定。这种方式更灵活，而且可以实现任意的排序准则：

 复制代码

```

1 set<int> s = {7, 3, 9};         // 定义集合并初始化3个元素
2
3 for(auto& x : s) {             // 范围循环输出元素
4     cout << x << ", ";         // 从小到大排序, 3,7,9
5 }
6
7 auto comp = [](auto a, auto b) // 定义一个lambda, 用来比较大小
8 {
9     return a > b;               // 定义大于关系
10 };
11
12 set<int, decltype(comp)> gs(comp) // 使用decltype得到lambda的类型
13
14 std::copy(begin(s), end(s),      // 拷贝算法, 拷贝数据
15            inserter(gs, gs.end())); // 使用插入迭代器
16
17 for(auto& x : gs) {             // 范围循环输出元素
18     cout << x << ", ";         // 从大到小排序, 9,7,3
19 }

```

除了**比较函数**这点，有序容器其实没有什么太多好说的，因为就这两个，选择起来很简单：**集合关系就用 set，关联数组就用 map。**

不过还是要再提醒你一点，因为有序容器在插入的时候会自动排序，所以就有隐含的插入排序成本，当数据量很大的时候，内部的位置查找、树旋转成本可能会比较高。

还有，如果你需要实时插入排序，那么选择 set/map 是没问题的。如果是非实时，那么最好还是用 vector，全部数据插入完成后再一次性排序，效果肯定会更好。

无序容器

有“有序容器”，那自然会有对应的“无序容器”了。这两类容器不仅在字面上，在其他方面也真的是完全对应。

无序容器也有四种，名字里也有 set 和 map，只是加上了 unordered（无序）前缀，分别是 unordered_set/unordered_multiset、unordered_map/unordered_multimap。

无序容器同样也是集合和关联数组，用法上与有序容器几乎是一样的，区别在于内部数据结构：**它不是红黑树，而是散列表**（也叫哈希表，hash table）。

因为它采用散列表存储数据，元素的位置取决于计算的散列值，没有规律可言，所以就是“无序”的，你也可以把它理解为“乱序”容器。

下面的代码简单示范了无序容器的操作，虽然接口与有序容器一样，但输出元素的顺序是不确定的乱序：

 复制代码

```
1 using map_type =                // 类型别名
2     unordered_map<int, string>;  // 使用无序关联数组
3
4 map_type dict;                  // 定义一个无序关联数组
5
6 dict[1] = "one";                // 添加三个元素
7 dict.emplace(2, "two");
8 dict[10] = "ten";
9
10 for(auto& x : dict) {          // 遍历输出
11     cout << x.first << ">"      // 顺序不确定
12         << x.second << ",";    // 既不是插入顺序，也不是大小序
13 }
```

无序容器虽然不要求顺序，但是对 key 的要求反而比有序容器更“苛刻”一些，拿 unordered_map 的声明来看一下：

 复制代码

```
1 template<
2     class Key,                  // 第一个模板参数是key类型
3     class T,                    // 第二个模板参数是元素类型
4     class Hash = std::hash<Key>, // 计算散列值的函数对象
```

```
5     class KeyEqual = std::equal_to<Key> // 相等比较函数
6 > class unordered_map;
```

它要求 key 具备两个条件，一是**可以计算 hash 值**，二是**能够执行相等比较操作**。第一个是因为散列表的要求，只有计算 hash 值才能放入散列表，第二个则是因为 hash 值可能会冲突，所以当 hash 值相同时，就要比较真正的 key 值。

与有序容器一样，要把自定义类型作为 key 放入无序容器，必须要实现这两个函数。

“==” 函数比较简单，可以用与 “<” 函数类似的方式，通过重载操作符来实现：

```
1 bool operator==(const Point& a, const Point& b)
2 {
3     return a.x == b.x;           // 自定义相等比较运算
4 }
```

 复制代码

散列函数就略麻烦一点，你可以用函数对象或者 lambda 表达式实现，内部最好调用标准的 std::hash 函数对象，而不要自己直接计算，否则很容易造成 hash 冲突：

```
1 auto hasher = [](const auto& p)    // 定义一个lambda表达式
2 {
3     return std::hash<int>()(p.x);  // 调用标准hash函数对象计算
4 };
5
```

 复制代码

有了相等函数和散列函数，自定义类型也就可以放进无序容器了：

```
1 unordered_set<Point, decltype(hasher)> s(10, hasher);
2
3 s.emplace(7);
4 s.emplace(3);
5
```

 复制代码

有序容器和无序容器的接口基本一样，这两者该如何选择呢？

其实看数据结构就清楚了，**如果只想要单纯的集合、字典，没有排序需求，就应该用无序容器，没有比较排序的成本，它的速度就会非常快。**

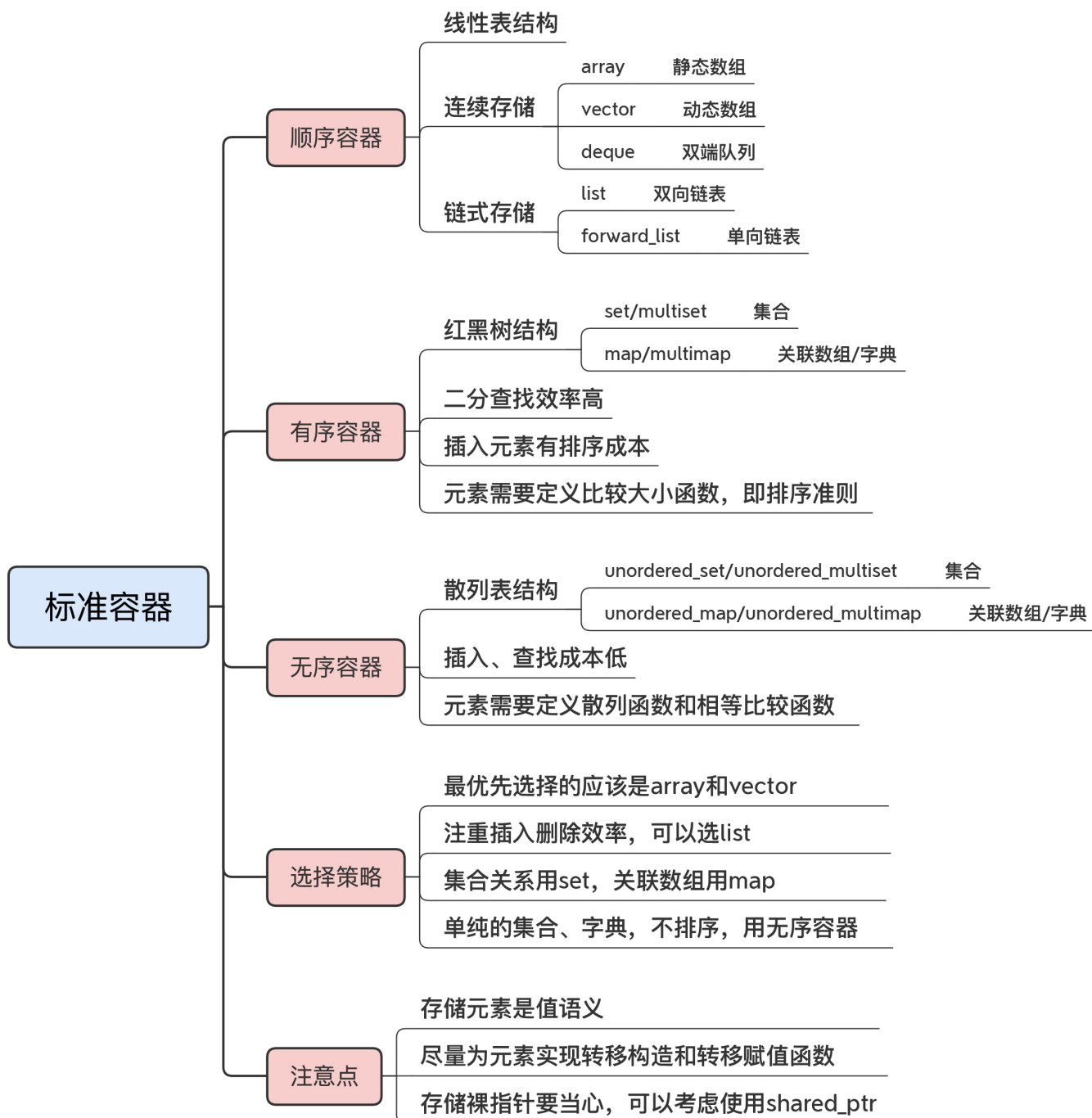
小结

好了，今天我从数据结构的角度全面介绍了 C++ 标准库里的各种容器，只要你了解这些容器的基本特性，知道内部结构上的优缺点，今后在写程序的时候，也就不会再犯“选择困难症”了。

判断容器是否合适的基本依据是“**不要有多余的操作**”，也就是说，不要为不需要的功能付出代价。比如，只在末尾添加元素，就不要用 deque/list；只想快速查找元素，不用排序，就应该选 unordered_set。

再简单小结一下今天的内容：

1. 标准容器可以分为三大类，即顺序容器、有序容器和无序容器；
2. 所有容器中最优先选择的应该是 array 和 vector，它们的速度最快，开销最低；
3. list 是链表结构，插入删除的效率高，但查找效率低；
4. 有序容器是红黑树结构，对 key 自动排序，查找效率高，但有插入成本；
5. 无序容器是散列表结构，由 hash 值计算存储位置，查找和插入的成本都很低；
6. 有序容器和无序容器都属于关联容器，元素有 key 的概念，操作元素实际上是在操作 key，所以要定义对 key 的比较函数或者散列函数。



我再教你一个使用这些容器的小技巧，就是**多利用类型别名，而不要“写死”容器定义**。因为容器的大部分接口是相同的，所以只要变动别名定义，就能够随意改换不同的容器，对于开发、测试都非常方便。

课下作业

最后是课下作业时间，给你留两个思考题：

1. 试着用自己的语言说一下这些容器的优点、缺点和区别。

2. 你最喜欢、最常用的是哪种容器？为什么？

欢迎在留言区写下你的思考和答案，如果觉得今天的内容对你有所帮助，也欢迎分享给你的朋友。我们下节课见。

课外小贴士

1. “算法 + 数据结构 = 程序” (Algorithms + Data Structures = Programs) 出自Pascal之父、图灵奖获得者Niklaus Wirth。
2. 评判数据结构的基本指标是空间复杂度和时间复杂度，后者就是我们常说的“Big O”。
3. 有了vector，就千万不要再使用“new/delete”来创建动态数组了。
4. 散列表理论上的时间复杂度是 $O(1)$ ，也就是常数，比红黑树的 $O(\log N)$ 要快。但要注意，如果对元素里的大量数据计算hash，这个常数可能会很大，也许会超过 $O(\log N)$ 。
5. 有序容器和无序容器里有“等价” (equivalent) 与“相等” (equality) 的概念，“等价”的意思是“ $!(x < y) \ \&\& \ !(x > y)$ ”，而“相等”的意思是“ $==$ ”。“等价”基于次序关系，对象不一定相同，而“相等”是

两个对象真正相同。

6. 标准库里还有 `stack`、`queue`、`priority_queue` 三个“容器适配器”，它们并非真正的容器，而是由其他容器（通常是 `deque` 或 `vector`）“适配”而实现的，使用接口上有变化，而内部结构相同。



更多课程推荐

MySQL 实战 45 讲

从原理到实战，丁奇带你搞懂 MySQL

林晓斌

网名丁奇
前阿里资深技术专家



涨价倒计时

今日秒杀 **¥79**，6月13日涨价至 **¥129**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 11 | 一枝独秀的字符串：C++也能处理文本？

下一篇 13 | 五花八门的算法：不要再手写for循环了

精选留言 (13)

写留言



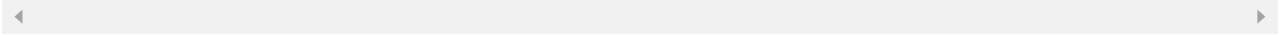
EncodedStar

2020-06-02

这节课把容器简直讲活了，最后的小技巧很实用，nice

展开 ▾

作者回复: thanks。



👍 3

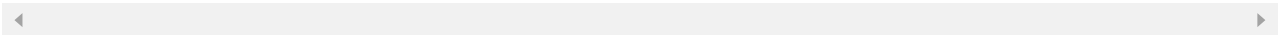


this_is_for_u

2020-06-02

关于vector扩容机制，建议加个平台前置条件，windows和linux系统的stl vector平台的扩容倍数是不一样的，而且移动端平台也有好多种stl，有gnustl和c++sharedstl等，不清楚具体实现，可能也会有所区别！

作者回复: 感谢补充，在Linux下习惯了，不太关注其他平台。



👍 3

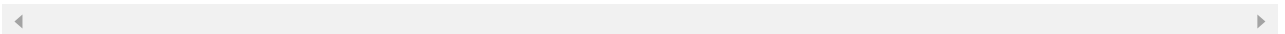


Eason Tai

2020-06-02

有一说一，配合c++ prime和自己手敲来看专栏，互相验证，真的挺舒服的。

作者回复: 很好的学习方法，nice。



👍 2



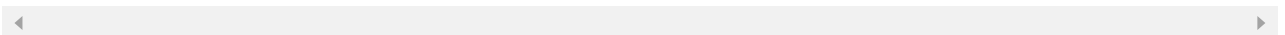
逸清

2020-06-02

老师，C++有什么比较常用的网络应用框架或库，最近要部署一个应用服务让其他人来调用，想用C++来实现

作者回复: 后面会专门讲。

简单来说就是boost.asio，即将进入标准库，但它还是很复杂的，用好了不容易。



👍 2



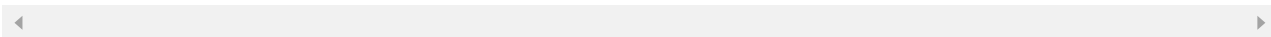
木瓜777
2020-06-03

如果使用unordered_map，对自定义的结构，例如 struct Point {float x;float y}，该如何实现hash？对多个float 有没有好的hash方法？

展开 ▾

作者回复: 可以使用标准库里的std::hash函数对类成员逐个hash。

但浮点数不建议用hash，它不精确。



1

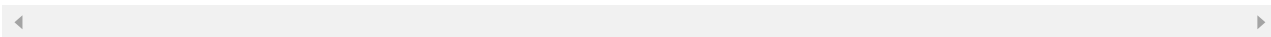


范闲
2020-06-02

容器里还有个重要的概念是迭代器。迭代器会失效，这点需要注意的。Array是cpp11以后才有的容器。

展开 ▾

作者回复: 后面的算法会简单讲一下迭代器，它的概念和用法要难理解一些，我的建议是尽量不要直接使用迭代器，用range-for、算法等把它隐藏起来。



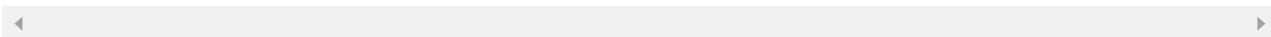
1



无为而立
2020-06-02

顺便复习了下，decltype哈哈，感觉把它当成函数更好理解

作者回复: 说得对，也可以把它理解成是编译阶段的函数，计算表达式的类型。



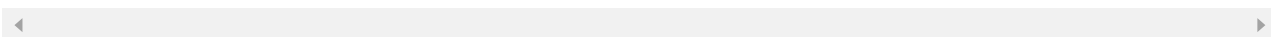
1



木瓜777
2020-06-02

您好！ 有时候经常会混用 map和unordered_map，这两者在存储大量数据的时候？谁效率更高？ 我看过几篇文章说，如果设计不好哈希值，unordered_map的查找效率会很低

作者回复: 在大量数据的时候最好用测试数据来说话，通常来说是无序容器效率高，标准库的hash函数适用性还是比较好的，当然也可以在模板参数里自己换。





java2c++

2020-06-08

问题1:所有的容器都是为了用来存放元素，理论上直接用数组就可以了，但是增删改查的效率未必如你所愿了。所以标准库又搞了那么多容器是为了满足各个不同的使用场景。增删改查效率最高的underd_set，时间复杂度是O（1），不过它是无序的，另外不能按下标查询。其次是红黑树，跳表，时间复杂度是O（logN）。 ...
展开

作者回复: 说的很好。

注重查询效率通常就会选择无序容器，但还有其他很多时候容器注重的是存储，所以就可以选择array、vector，等全存完了之后再排序查找。



Ryan24G

2020-06-05

老师，无序容器中，查找key使用的是什么算法，查询效率和有序容器比起来怎么样？

作者回复: C++标准只是规定了无序容器的基本概念和性能要求，而没有规定容器的具体实现，所以各家编译器和标准库可以自由发挥，采用不同的算法，不是统一的。

因为无序容器是散列表，所以查找效率通常来说要比红黑树来的高，因为只需要计算一次hash值，通常就可以直接定位到元素，比红黑树的多次查找要快很多。

但这个也只是理论上的说法，实际效果怎么样还是要做个测试。



群

2020-06-05

文中说key必须具备两个条件,其中第二个条件,"第二个则是因为 hash 值可能会冲突,所以当 hash 值相同时,就要比较真正的 key 值",当hash值一样时,直接把新添加的元素添加到hash值相等的队列后边不就行了吗?为什么再比较key值呢?
这里,比较key值有什么用处吗?比较了之后可以用来做什么呢?没明白这里的意思,老师可...

展开 ▾

作者回复: 举个例子吧, 比如说用string作为key, abc和cba, 两者的hash值是相同的, 都放进了一个开链表里, 但这两个元素显然是不同的, 真正要找cba就必须在这个链表里取比较查找。

关键在于hash值相等不代表key就一定相等, 因为hash值只是一个映射, 还必须去比较真正的实体。

◀ ▶



有学识的兔子

2020-06-02

结合课程内容和debug test cases, 对容器有了比较好的理解, 也对顺序/有序/无序容器使用了面向对象的抽象和封装, 提供了可供调用的类接口以及隐藏了数组 链表 集合 散列树等一系列数据结构的实现细节, 更加便利的驾驭较为复杂的数据结构。

展开 ▾

作者回复: 把容器和数据结构知识结合起来, 就能够用好容器了。

不过C++的容器也为标准数据结构增加了一些额外的特性, 比如emplace、迭代器等等, 需要再看接口文档。

◀ ▶



大豆

2020-06-02

老师, 可不可以这样理解。哈希表其实就是一个数组, 是通过对hash值做一些计算来得出元素的下标, 然后再取出元素。

如果hash冲突则下标顺延或者在当前下标处加一个链表。

展开 ▾

作者回复: 是的, 可以再看一些经典的数据结构书籍或者教材。

◀ ▶

